



fig. 1
(PRIOR ART)

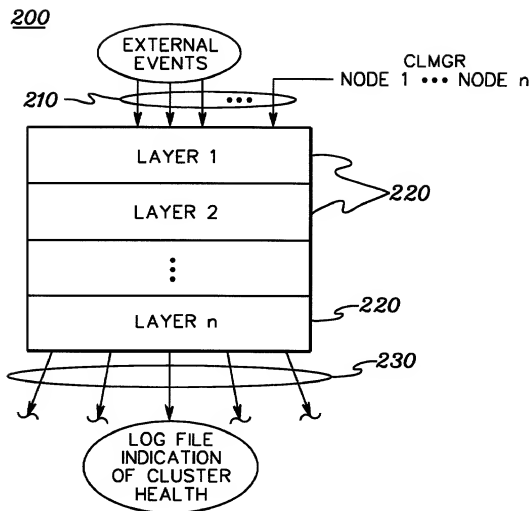


fig. 2

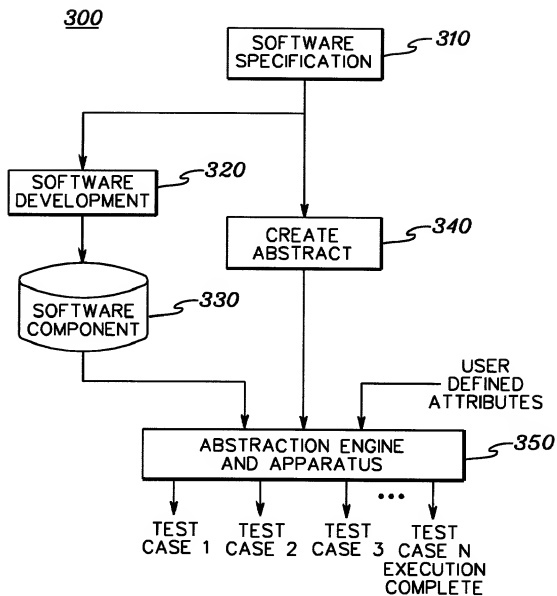


fig. 3

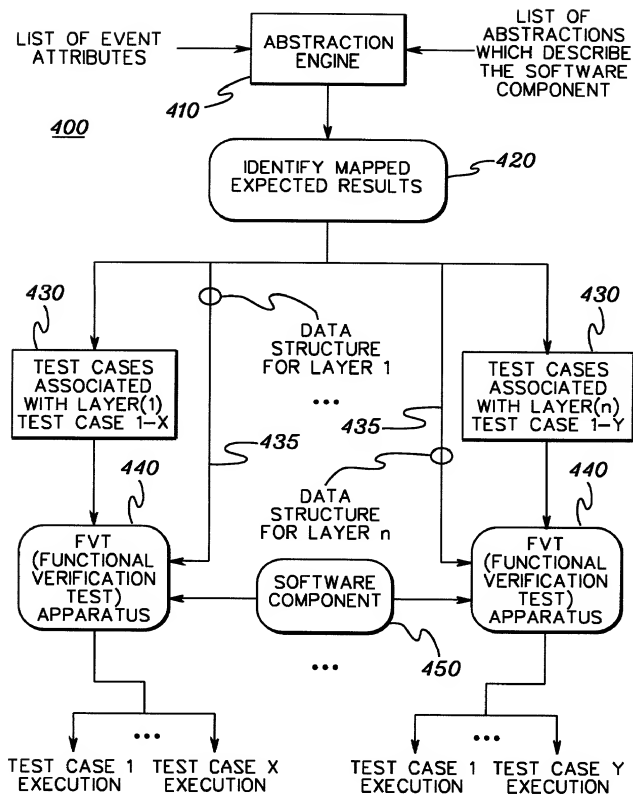


fig. 4

500

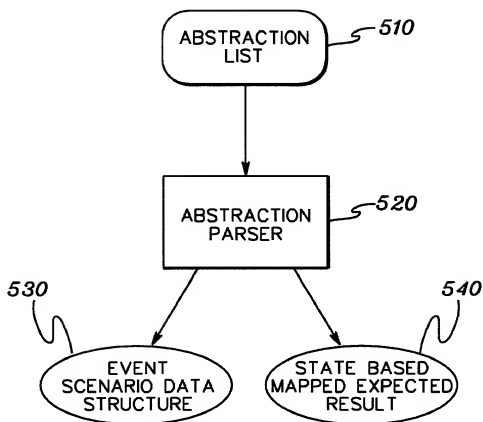


fig. 5

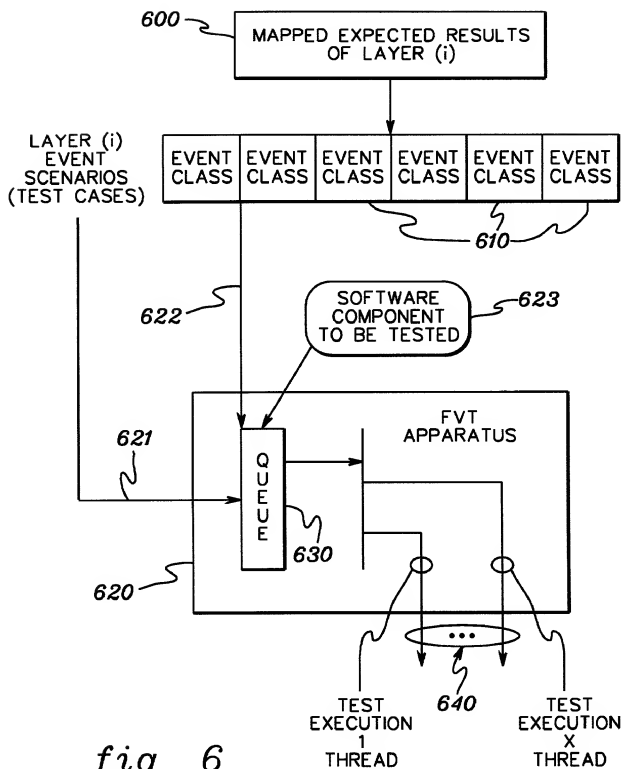


fig. 6

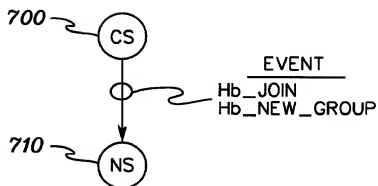


fig. 7

800

LAYER 2
ABSTRACTION FILE

⋮	
⋮	
CS: NODE ADAPTER UP	NS: AMG_STATE = STABLE
⋮	
⋮	

fig. 8

900

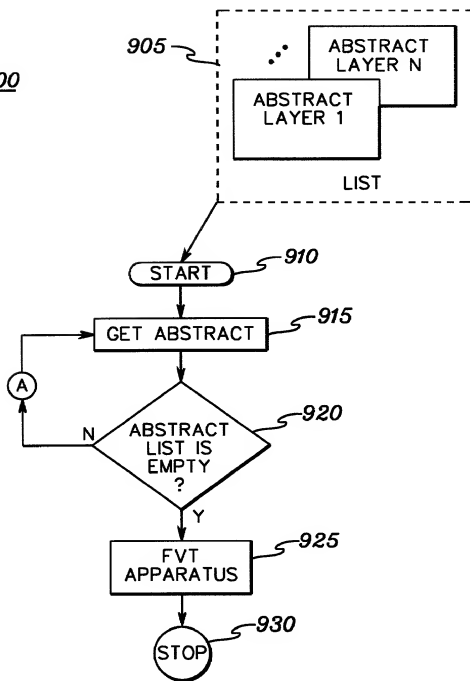


fig. 9A

09919753.000101



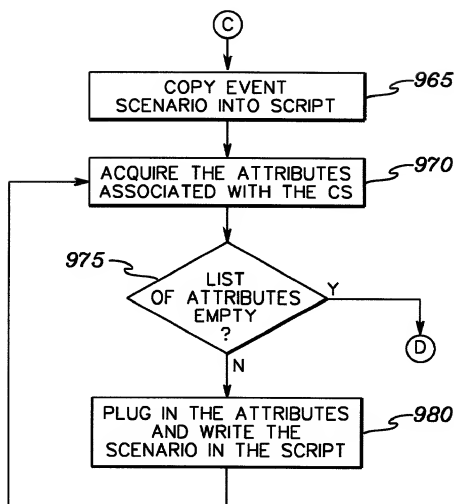


fig. 9C

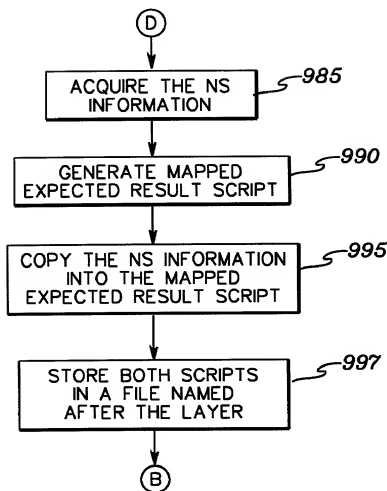


fig. 9D

PSUEDO_CODE.TXT

PSEUDOCODE FOR THE ABSTRACTION ENGINE AND POV (POINT OF VERIFICATION)
EACH LAYER HAS ITS OWN ABSTRACT FILE.

OPEN FILE. FOR EACH LINE DO THE FOLLOWING

READ CS: TOKEN (CURRENT STATE)
READ NAME AND ACQUIRE THE DATABASE (FILE CONTAINING THE DETAILED
EVENT SCENARIO, THAT WILL TAKE YOU TO THE "NEXT STATE")
(THIS INFORMATION IS THE SYNTAX AS DESCRIBED IN THE SYSTEM DESIGN)

AN EXAMPLE IS Hb_JOIN <ADAPTER> WHERE ADAPTER IS A VARIABLE
WHICH WILL BE FILLED IN LATER

CREATE A TEMPLATE KSHELL SCRIPT,
COPY THE EVENT SCENARIO FROM THE DATABASE.
PLUG IN THE ATTRIBUTES AND CREATE AN EVENT SCRIPT
REPEAT UNTIL YOU HAVE REACHED THE END OF THE ATTRIBUTE LIST

EXAMPLE Hb_JOIN end WHERE end IS AN ADAPTER TAKEN FROM THE
ATTRIBUTES LIST WHERE EACH SCENARIO CONSTITUTES A TEST CASE

READ NS: TOKEN

CREATE PERL SCRIPT TEMPLATE, NAME IT AFTER THE NS TOKEN NAME
EXAMPLE <AMG_STATE STABLE>

MODIFY PERL SCRIPT WITH THE EVENT SCRIPT(S)
WHICH CONTAINS THE DETAILED SCENARIO OF WHAT THE "NEXT STATE" IS
(THIS INFORMATION IS WHAT SHOULD BE CONTAINED IN THE SYSTEM
DESIGN DOCUMENT. ALSO THIS INFORMATION WAS USED AS PART OF THE
REVIEW BY THE TESTER, AND THE COMPONENT DEVELOPER)

CREATE THE EVENT CLASS (MAPPED EXPECTED RESULTS) USING THE
DETAILED INFORMATION ABOVE, AND THE ATTRIBUTES.

STORE THE TESTCASE PERL SCRIPT AND EVENT CLASS, IN
DATABASE/FILE NAMED AFTER THE LAYER.

fig. 10

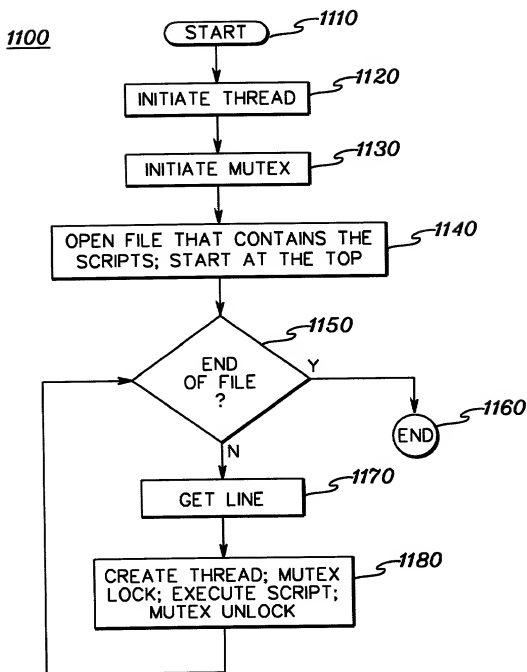


fig. 11

```

/* PseudoCode for fvt apparatus */
#include <pthread.h>
#include <stdio.h>

/* This is the initial thread routine */
void* compute_thread(void*);

/* This is the lock for thread synchronization */
pthread_mutex_t my_sync;

/* This is the condition variable for task order control */
pthread_cond_t rx;

/* This is the Boolean */
int thread_done = FALSE;

main()
{
    /* This is data describing the thread created */
    pthread_t tid;
    pthread_attr_t attr;

    /* Start of executable */

    /* Initialize the thread attributes */
    pthread_attr_init(&attr);

    /* initialize the mutex (default attributes) */
    pthread_mutex_init(&my_sync, NULL);

    /* initialize the condition variable (default attributes) */
    pthread_cond_init(&rx, NULL);

```

fig. 12A

```

/* Create another thread. The Thread ID is returned in &tid */
/* The last parameter is passed to the thread function */
while (the file containing the testcases for a particular
layer is not empty do the following)
{
pthread_create (&tid, &attr, compute_thread, invoke_perlscript);

/* wait until the thread does its work */
pthread_mutex_lock(&my_sync);
while (!thread_done) pthread_cond_wait (&rx,&my_sync);

/* When we get here, the thread has been executed */
printf(thread);
printf("\n");
pthread_mutex_unlock(&my_sync);
exit(0);
/* end of do while
/* end of main routine

/* The thread to be run by create_thread */
void* compute_thread(void* invoke_perlscript)
{
/* Lock the mutex when its our turn */
pthread_mutex_lock(&my_sync);

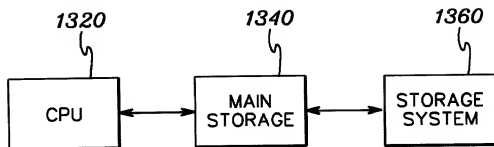
invoke_perlscript
/* set the predicate and signal the other thread */
thread_done = TRUE;
pthread_cond_signal(&my_sync);
pthread_mutex_unlock(&my_sync);

return;
}

```

fig. 12B

00019753.000101
TOTAL 26660



1300

fig. 13

00919753 030104
101020 22/6160